A Crash Course on Python

What is Linux?

- Linux is an open-source OS designed to run on many different devices (mobile, desktop, etc). Android is based on Linux.
- Course uses Ubuntu, one of the more popular distributions (versions) of Linux.
• Navigate around using Terminal (same as mac).
- Terminal is a text-based interface for sending commands to a computer.
- Create new files using the touch command: touch myfile.txt
- The ~ means home folder or default directory.
- Other commands:
• ls — list all filed in the current working directory
• pwd — display current working directory.
• cd — navigate
• cat — print out contents of a file.
• rm — remove/delete a file
• mkdir — create a new directory/folder
• rmdir — remove/delete a directory

Intro to Python (Print and Data Types)

- Strings and Booleans.
• String literal: Used alone in a print statement.
• String variables: Binding a string to a name (>>> myString = “Hello”)
• Boolean: True and False
- 5 = 3 is False.
- 3 > -1 is True.
- Print statements.
• Extra: Difference between print and return.
- From CS 61A: “Return statements allow the programmer to return a value from a function. Print statements on the other hand just print what you want to the screen and return None. It doesn't allow you to actually use the value that you printed.”

- Integers
• Integers are numbers without decimals: 12, -1, 0, 3
• Floats are numbers with decimals: 0.999, 1.0, 0.5

Conditional Statements

- Python supports relational operators including >, <, <=, >=, ==, and, or,
not.
1
• For now, we’ll just use the first 5.

```
>>> life = 3
>>> life -= 1
>>> life
2
>>> life += 1
>>> life
3
```

- Note: minus equals and plus equals are examples of special syntax in python
• They are used to increment variables
• var = var + 1 is the same as var += 1
• var = var - 1 is the same as var -= 1
- What if we try life -= 1 more time? (repeat until life gets to -1).
• But, we can’t have a negative life. We can use a conditional statement to do this.

```
>>> life = 3
>>> if life > 0:
 life -=1
>>> if life > 0:
 life -=1
>>> if life > 0:
 life -= 1
>>> if life > 0:
 life -=1
>>> else:
 print(“The player ran out of life!”)
>>> print(life)
```

- We can use elif (short for else if) to do something when the first if statement isn’t true, but before the else statement.
- Make the program print out a sentence if life is equal to zero.

```
>>> life = 3
>>> if life > 0:
 life -=1
>>> if life > 0:
 life -=1
>>> if life > 0:
 life -= 1
>>> if life > 0:
 life -=1
>>> elif life == 0:
 print(“The player has zero life.”)
>>> else:
 print(“The player ran out of life!”)
>>> print(life)
```

- Note: Difference between = and ==.

• life = 3 is assignment. life == 3 is a boolean value (“Is life equal to 3?”)

- Note: What would happen if we changed the code to:

```
>>> life = 3

>>> if life > 0:
 life -=1
>>> elif life > 0:
 life -=1
>>> elif life > 0:
 life -= 1
>>> elif life > 0:
 life -=1
>>> elif life == 0:
 print(“The player has zero life.”)
>>> else:
 print(“The player ran out of life!”)
>>> print(life)
```

- Back to and, or, not.

• These are boolean operators.

• Use and to check if multiple conditions are true.

- 3 > 5 and 5 > 3 is False.

• Use or to check if at least one condition is true.

- 3 > 5 or 5 > 3 is True.

• Use not to check if something is false.

- This keyword is commonly used in games when you want the game to keep

running if it isn’t won or over.
- not 3 > 5 is True. not 5 > 3 is False.
• Try it yourself in the terminal!
- Summary: Use if, elif, and else statements to do different things based on certain Conditions.

Loops

- In programming, loops allow you to repeatedly execute a block of code.
- Sometimes, we need to execute a block of code an unknown or non-specific amount of times, say until a certain condition is met. This kind of loop is called a while loop.
• How many guesses will it take to guess how many leaves are in a tree?
- A control variable is used to set when the loop does and doesn’t run.
• Here, guessed is our control variable. The loop repeats until the condition we set is met. Use the control variable in the conditional statement of your loop.
3
Queue’s Artificial Intelligence and Machine Learning Class

```
guessed = False
while not guessed:
 # YOUR CODE HERE
 guess = input(“Guess a number: “)
 if int(guess) == 14:
 guessed = True
 print(“Correct answer:”, guess)
 print(“Guessed?”, guessed)
```

• When the while loop concludes, we can print out the new value of guess and guessed.
- A for loop is different from a while loop because it is used to repeatedly execute a block of code a finite/known amount of times.
• For example, adding 1 to every element in the list [1, 2, 3, 4, 5].
- For loops require the same 3 components as while loops:
• A control variable
• A conditional statement
• A loop body.
- Say we want to do something to all 14 leaves in the tree from before.
• Use the built-in range function in the conditional statement of the loop.
• Each time the loop runs, it will add 1 to the count variable x, and the loop terminates when x == number_of_leaves.

```
number_of_leaves = 14
for x in range(number_of_leaves):
print(“A leaf fell to the ground.”)
print(str(x) + “ leaves have fallen.”)
print(“All the leaves fell.”)
```

Random Numbers + Using Python Libraries

- Libraries are collections of remade code that you can import and use in your code.
- Often you will need to generate a random number in python (simulate flipping a coin, rolling a die, etc.) To do so, we need to import the random module using the random keyword.
• The boundaries for the integers you want to generate are inclusive on both ends.

```
import random
 print(“Rolling a die…”)
 print(random.randint(1, 6))
```

Functions

- Functions are a powerful abstraction technique that allows you to reuse and simplify code.
- Function declaration statements are as follows:

```
def my_function(arguments):
 # CODE TO RUN
```

• Arguments (kind of like a variable) are a way for you to provide more information to a function. The function uses the argument while it runs.
• A function can have any arbitrary number of arguments (including 0!). Generally, try to avoid using more than 8, otherwise it gets too messy.
- Suppose we want to write a function to generate a random number.

```
def random_number():
 rand = random.randrange(0, 2)
 print(rand)
```

- Then, we call our new function as follows:

```
 >>> random_number()
```

- Functions can also return a value. There is an important difference between print and return.

```
def random_number():
 rand = random.randrange(0, 2)
 return rand
 >>> a_number = random_number()
 >>> print(a_number)
```

- What if we want to determine the biggest possible number our random number function?

```
def random_number(max_num):
 rand = random.randrange(0, max_num)
 return rand
 >>> a_number = random_number(5)
 >>> print(a_number)
```

Lists

- A list is a data type that holds a collection of values. Lists can be composed of any type: strings, int, floats, and more.
- Lists are denoted using square brackets []

num_list = [91, 92, 93, 94, 95, 96]

- You can also use square brackets to access items in a list.
• Lists are zero-indexed!
- Here’s why: Dijkstra's Why numbering should start at zero
• Thus, num_list[0] = 91, and num_list[1] = 92
- We can sort lists by using the built-in function .sort
• We call it as follows:

```
 >>> my_list = [99, 324, 139, 2]
 >>> my_list.sort()
 >>> my_list
 [2, 99, 139, 324]
```

• We can also sort lists of strings (alphabetic order).
- We can add things to a list by using .append which adds elements to the end and .insert which adds elements in a given spot.

```
>>> my_list = [93, 4, 6, 1]
>>> my_list.append(5)
>>> my_list
[93, 4, 6, 1, 5]
>>> my_list.insert(0, 100)
>>> my_list
```

```
[100, 93, 4, 6, 1, 5]
>>> my_list.insert(1, 99)
>>> my_list
[100, 99, 93, 4, 6, 1, 5]
```

- Lists also have more special functions, feel free to experiment with these on your own and see what they do!

• pop()
• pop(index)
• remove(elem)